CLUSTERED COMPUTING WITH NETLOGO AND REPAST J: BEYOND CHEWING GUM AND DUCT TAPE

M.T.K. KOEHLER* and B.F. TIVNAN, The MITRE Corporation, McLean, VA
S. UPTON, Referentia Systems, Inc., Tampa, FL

ABSTRACT

We have developed a methodology to run NetLogo in an automated fashion in a cluster computing environment or on a single machine, varying both parameter values and/or random seeds. We utilize the Java application program interface (API) of NetLogo, as well as those of Condor, and three custom software programs: a NetLogo XML parser, XStudy, and OldMcData (OMD). This paper describes how to set up a NetLogo program to run in this environment, use the XML parser and XStudy to create the other necessary files, and then use OMD to perform the runs and collect the output data files. The software currently supports automation of statistical experimental designs (such as a nearly orthogonal Latin hypercube), evolutionary algorithms (utilizing a user-defined, potentially very complex fitness function), and full factorial parameter sweeps. All of the software discussed in the paper will be made available to the research community. The current system represents a fully functional prototype, although additional development work is ongoing to improve the robustness and accessibility of the system. The system described herein will also work with Repast J. However, the system requires writing some additional Java code that is specific to the Repast J model. It is our hope in the future to make the current system work with Repast J more seamlessly. We hope the release of the software and methodology will be seen as an invitation to collaborate to improve the system as a whole and enhance its utility to the vibrant modeling communities of NetLogo and Repast J.

Keywords: NetLogo, Repast, cluster computing

INTRODUCTION

Many agent-based models have some inherent randomness associated with them. This means there will likely be a distribution associated with output from the models. One must be cognizant of where on the distribution of possible outcomes a particular model run may lie. This is difficult to know a priori, however. Therefore, it can be particularly useful to run agent-based models many times, varying not only model parameter values but also random seeds. Done manually, however, this task is tedious in the extreme. This paper will discuss a methodology to run NetLogo many times in an automated fashion in a cluster computing environment that is more flexible than, although not as user friendly as, NetLogo's internal BehaviorSpaces.

The work described in this paper is the outgrowth of work started as a part of Project Albert known as data farming and operational synthesis; see Brandstein (1998) and Horne (2001) for a more complete discussion. One of the many foci of Project Albert was the creation of a data

* *Corresponding author address:* Matthew Koehler, The MITRE Corporation, 7515 Colshire Dr., Mail Stop H305, McLean, VA 22102; e-mail: mkoehler@mitre.org.

farming environment (DFE); this is a cluster computing environment designed to run simple, stochastic models many times (tens of thousands to millions) as a way to explore the dynamics of the model and understand the possible outcomes, especially potential outlier events; see Barry (2004a,b) for a discussion of the importance and utility of understanding outlier events. This was done with an eye toward decision support, so the system was made as easy to use as possible in the hope that it would be utilized by decision makers and subject matter experts. The models that are currently part of the DFE include ISAAC, MANA, Pythagoras, Socrates, PAX, and NetLogo. ISAAC, MANA, Pythagoras, and Socrates are all agent-based combat models. PAX is an agent-based peace-support, peace-keeping model. NetLogo is a general-purpose agent-based modeling environment (Wilensky 1999). The incorporation of NetLogo represents a major step forward for the generalization of the DFE, making it useful not only to military analysts but also to the greater academic and analytic communities.

The DFE is based on XML input and study definition files. This presented some challenges with respect to NetLogo, since it does not use XML for its input files. We overcame this by creating a few standards for the NetLogo program and a parser that breaks the NetLogo program into a series of XML blocks on the basis of each part's functionality. This paper briefly describes the major steps necessary to "data farm" a NetLogo model.

GENERAL FLOW OF THE DFE WITH NETLOGO

The general flow of the system is as follows: (1) create a NetLogo model following a set of conventions; (2) parse the NetLogo file into an input XML file; (3) by using the XStudy tool, pick the sliders, choosers, or switches that will be varied during the runs; (4) use OMD and Condor to kick off the runs and collect the data (this can de done on a single machine or multiple machines). All of the software is written in Java and should work on any machine with a Java Virtual Machine. This system, although it is not perfected, is robust enough to handle the pressure of workshop demands, including thousands of runs done remotely on clusters in different countries. We have successfully run Netlogo in two different cluster computing environments: the Maui High Performance Computing Center and the Singapore Defense Science Organization. The system is capable of handling any sort of experimental design, from full factorial to nearly orthogonal Latin hypercube (NOLH). Furthermore, OMD has post-processing capabilities that can be used with evolutionary programming algorithms and other types of user-defined algorithms to create a more dynamic study.

In the following discussion, we examine the conventions needed to put together a NetLogo program, and we provide general instructions for using the other software used for the multiple runs; however, we assume the reader is familiar with Condor. The software discussed in this paper is, or soon will be, available on SourceForge. Alternatively, the software is available from the authors. Condor is available from its developers at http://www.cs.wisc.edu/condor/. NetLogo is available from its developers at http://ccl.northwestern.edu/netlogo/.

SETTING UP THE NETLOGO MODEL

The current system requires certain features within the NetLogo model. These requirements, which are discussed below, have minimal impact on the structure of the program or the speed of execution and are designed to allow an external Java program to start the model, set parameter values (sliders, choosers, and switches), start and end a run, and collect output data (both end-of-run and time-series data). In general, the wrapper starts NetLogo and loads the model, and then it tells NetLogo to iterate a certain number of times. At the end of the requisite number of iterations, output data are collected, and the NetLogo run is terminated.

Global Variables

The model needs three global variables: `stopped`, `filename`, and `clock`. These are used by the external program to run NetLogo, keep track of output data, and allow the modelers to control the behavior of their model separately from the Java wrapper.

Setup

First, the NetLogo model must have a procedure called `setup` to instantiate the model and to prepare the output files. At a minimum, it needs the following lines of code:

```
to setup
set clock 0
set stopped false
setup-file
end
```

Every time the model is run, it will be in a newly started instantiation of NetLogo; therefore, you are not required to set variables (unless they need to be something other than zero). However, you may want to clear values and set others so that you will know exactly how the model is starting up. If you do clear values, DO NOT use the command `clear-all` or `ca`. If you want to clear values, use commands such as `clear-turtles`, `clear-patches`, `clear-all-plots`, or `clear-output`; then manually set the variables. If you use `clear-all`, you will set the variable `filename` to 0. This will cause problems later on, when the output from all the runs is collected, because all the files will have the same name. The batch version of NetLogo is run by a Java program that will set certain parameters; among them is `filename`. Once NetLogo is started, the Java program will call the `setup` procedure. If `setup` then resets the value of `filename`, Condor and OMD will have trouble keeping track of the output files because all the files will have the same name. A more comprehensive version of the setup procedure that includes resetting of values is shown below:

```
to setup
ct
cp
clear-output
clear-all-plots
;; manually set all variables
```

```
set clock 0
set stopped false
setup-file
end
```

The setup-file procedure is very short and could be called from within the setup procedure. It is recommended to keep them separate for clarity. A sample of this procedure is shown below:

```
to setup-file
 ifelse filename = 0
 [file-open “Your_BackUp_Name_Here.csv”]
 [file-open filename]
end
```

This procedure allows you to run the NetLogo program inside the cluster computing environment or in the standard NetLogo program for testing purposes. This works because it checks to see if the variable `filename` has been set by the Java wrapper program. If it has not been set by the Java wrapper, it will open a default file of your choosing.

Go

All models must also have a `go` procedure. The `go` procedure is a little different than the usual NetLogo program. First of all, the procedure must be called “`go`.” Second, the wrapper runs the NetLogo program by asking it to step a certain number of times. Because of this structure, it is important to “protect” your runtime code by nesting it inside an `if` statement that returns true if stopped is false. Sample code for the `go` procedure is given below:

```
to go
set clock clock + 1
if not stopped
  [
  ;;runtime code goes in here

  if ‘stop condition is true’
[do-file-print close-files set stopped true]
  ]
end
```

By nesting the runtime code inside the `if` statement, the wrapper can run the model any number of times without any potential damage to the output after the stop condition is met. For example, if you have set up the wrapper to run your model 6,000 times, but you have a stop condition that is triggered at time-step 3,500, the wrapper will continue to tell your model to step another 2,500 times. If you generate output at every time step and do not protect it, then you will end up with another 2,500 lines of output. Since your stop condition could be triggered at different times, it could be very difficult to fix your data after the run. It is also important to segregate any end-of-run printing procedures from the file close procedure. Once the wrapper is done stepping the NetLogo program, it will tell the program to `close-files`. Therefore, you

must have a procedure in your program that is called `close-files`. If this procedure includes anything other than file closing code, it may cause a problem, since it will be run any time files are closed. If you close files at any time that the stop condition for your model is true, then any other code will be run every time the wrapper steps your program once the stop condition is met. (This is not an issue if you protect your runtime procedures in the aforementioned `if` statement and make the `close-files` procedure exclusively devoted to closing files.) However, this does require that your model have a stop condition that will be triggered at least one time-step before the wrapper ends the run, because the wrapper will simply stop telling the program to step and then call the `close-files` procedure. Sample code for the `do-file-print` and `close-files` procedures are provided below:

```
to do-file-print
file-print “output goes here”
end

to close-files
file-close-all
end
```

Also, there is no post-processing currently associated with NetLogo runs, so if you want something in the output file, such as input parameters, you must write it there in the program (in something like the `do-file-print` procedure). This file will be a single line if you are only collecting end-of-run data. If, however, you are collecting time-series data, this file may be very large.

The above represents all the requisite code for a NetLogo program to set it up for cluster computing. Now, part of the utility of cluster computing is being able to run a model many times with different parameter values. The system we have developed can run NetLogo programs many times and change parameter values. However, the parameters that will change need to comport with a set of standards. First, they must be sliders, choosers, switches, etc., so they must therefore appear in the “Interface” tab of the NetLogo environment. Second, the parameters must not contain any special characters, like ?, %, $, or *. Third, they may be set to numeric values only — no strings. For example, a chooser with the values high, medium, and low would not be acceptable. The chooser should have values such as 1, 2, and 3, which could then be mapped to high, medium, and low in the procedural part of the NetLogo model. This does not preclude other parameters from taking on any values you wish or from having special characters in their name; these standards apply only to parameter values you wish to change in an automated fashion.

NETLOGO TO XML PARSER

Once the NetLogo model is completed, it is time to prepare the system for multiple runs. To do this, first create a new folder to use for file preparation and to collect the output created by the multiple runs. Place the NetLogo model in this folder, and make sure that it writes output to this folder. (NetLogo defaults to the user folder, which can cause problems later on, when OMD tries to collect output that is not in the right folder.) Next use NetLogo2XML to parse the NetLogo file into an XML format usable by XStudy. The process basically pulls the variables that are declared as sliders, choosers, and switches and places them within XML tags that

XStudy will recognize as variables that can be manipulated from one run to the next. To accomplish this conversion, place the NetLogo2XML.jar and the dom4j-1.5.jar into your working folder, then launch the parser from a command prompt within your working directory with the following commands:

```
java -cp NetLogo2XML.jar;dom4j-1.5.jar

albert.datafarm.netlogo.GenerateXMLScenario

Your-NetLogo-Model-Name.nlogo
```

with spaces in between the separate lines above. The parser then outputs an appropriate XML file based on your model name (e.g., Your-NetLogo-Model-Name.xml) into the same working directory. This file can then be used by XStudy to define your experimental design.

THE XSTUDY TOOL

Once the NetLogo XML file has been created, start XStudy by double clicking the `xpath.bat` file. Once XStudy is running, open the NetLogo XML file. Per Figure 1, in the box on the left, you will see all of the variables associated with the sliders, choosers, and switches of your model. Next, create a parameter group by clicking the "Add Param" button and typing in a name. You must have at least one group, but you do not need more than one group. To add a variable to the group, click the variable in the box on the left, then click the "Add Current Selection" button. Do this for all variables you wish to change during the run. Every variable that will take on different values during the runs will need to be in a different group. For example, if you have sliders associated with sight for three types of agent and you wish to vary them in the same way across all three groups, then place them in a single group — perhaps called vision. If there is another slider in your model for the number of obstacles within the environment and you wish to change it in a way that is dissimilar to the values for agent sight, then it will need to go in a different group — perhaps one called obstacles.

Per Figure 2, once you have created all of the necessary parameter groups, the Gridded tab can be used to create a simple, full factorial or gridded experimental design. In this tab, you will see a list of all the parameter groups you created. Simply enter in a minimum, maximum, and a delta for all the groups to create your study. This is very similar to the BehaviorSpace experimental setup found within NetLogo and the structure of the multi-keyword value definitions in Repast parameter files.

If you click the DOE radio button at the bottom of the Select Parameters tab (see Figure 1), the CSV_DOE tab becomes active. Figure 3 is a screen shot of that tab. This option allows you to create any study design you wish. It simply requires a CSV file of the values you wish to run for each parameter. The structure is simply this: columns are parameter groups created in the Select Parameters tab (order here is important, the left to right order of the columns in the CSV must be the same as the top to bottom order of the parameter groups), and each row is a run. To load the CSV file you created, browse to its location, tell XStudy how many lines to skip (if you have any header information in the file, tell XStudy which line has the labels in it), and then click Parse. Once that has been accomplished, click the cell in the Column

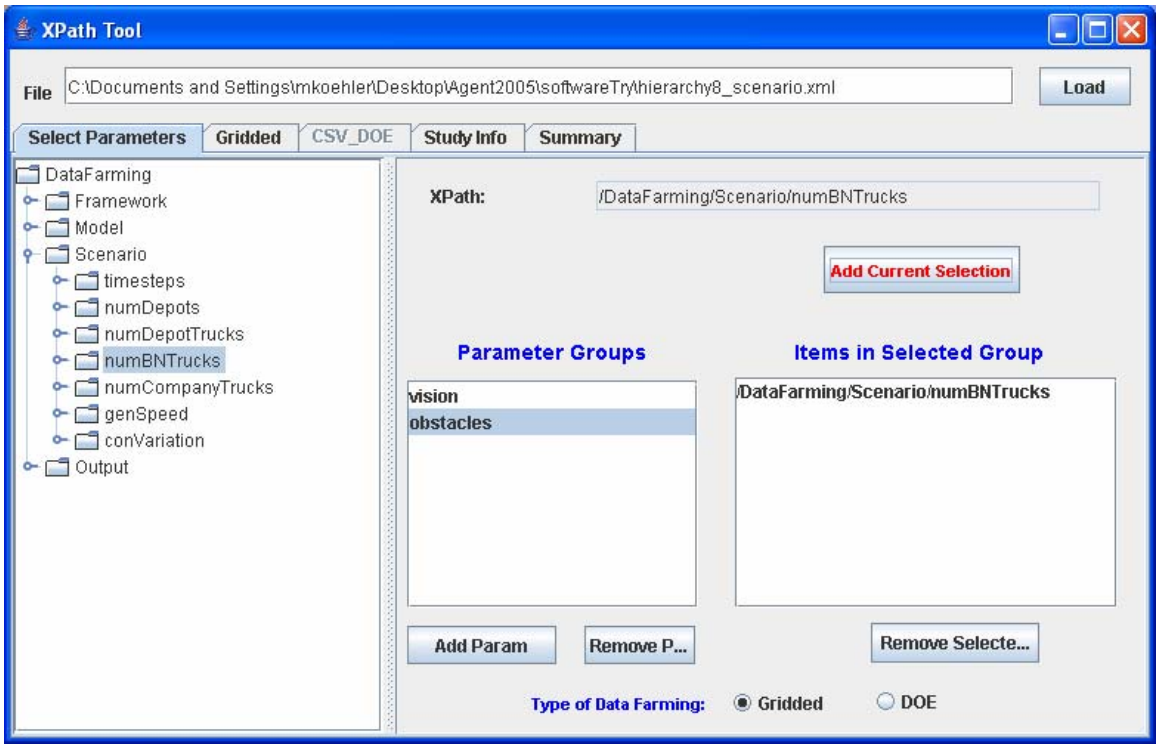

FIGURE 1 Select Parameters tab

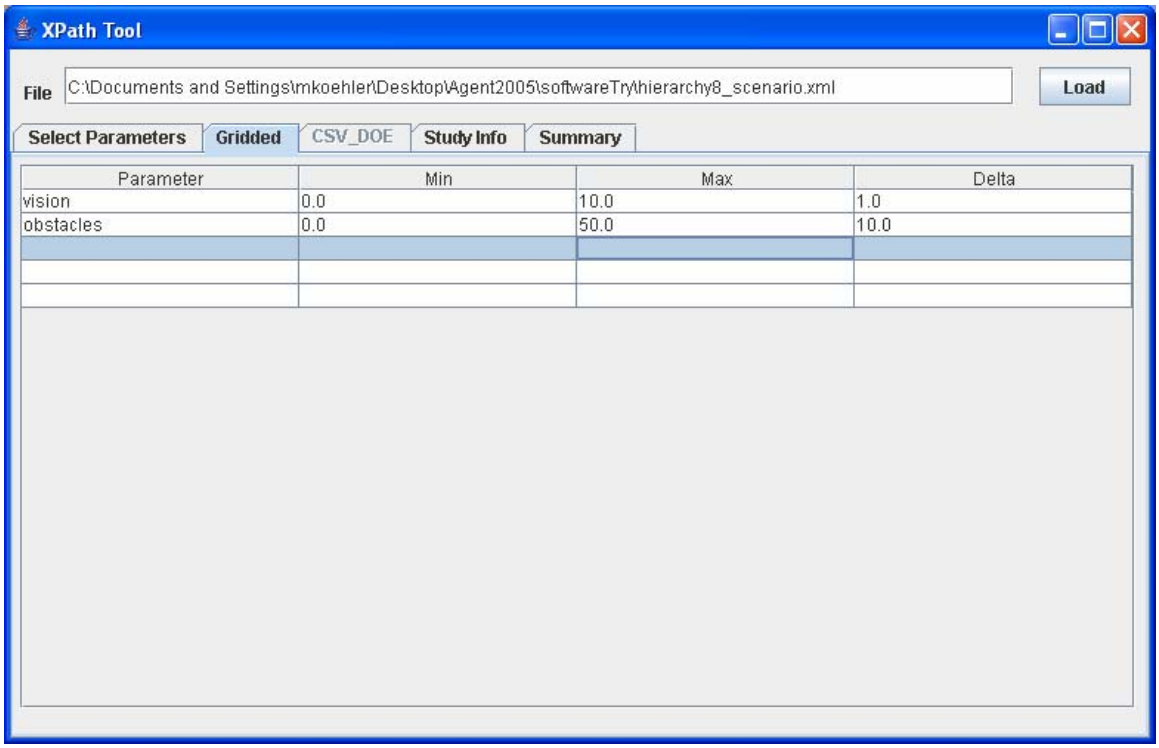

FIGURE 2 Gridded tab

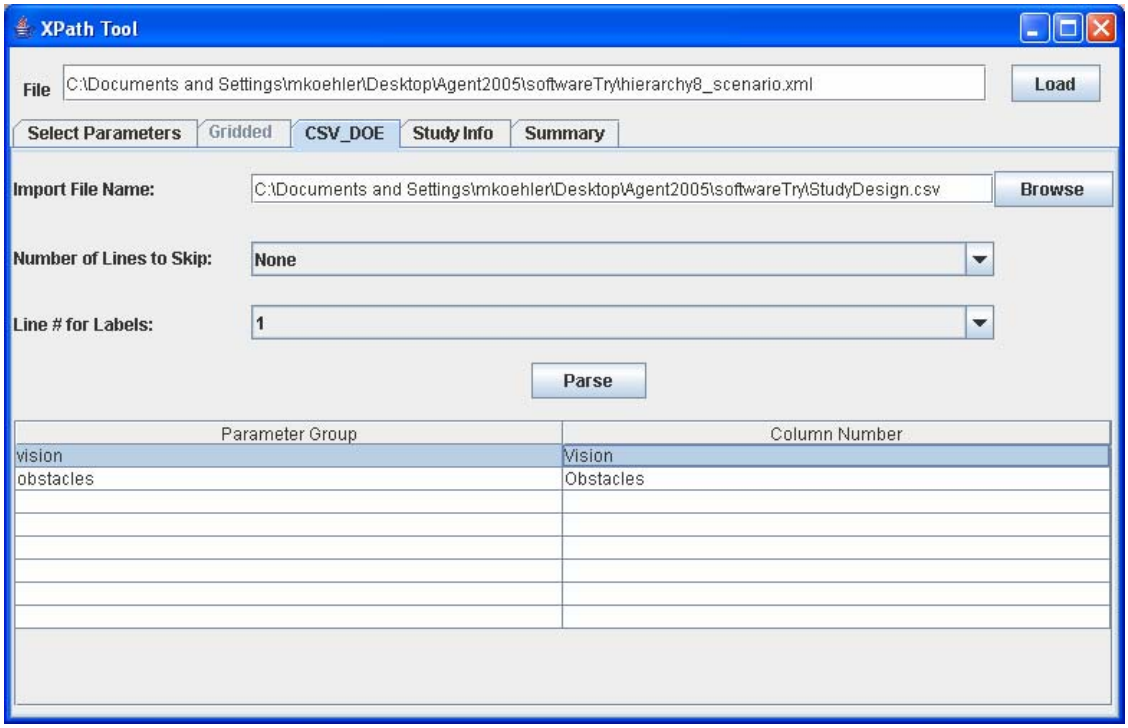

FIGURE 3 CSV_DOE tab

Number column adjacent to a parameter group and choose the appropriate column for the values from the drop-down menu.

It is in the CSV_DOE tab that you would create a more sophisticated study design, such as NOLH. Software to generate a NOLH study design in an Excel spreadsheet, as well as a wealth of other information, is available from the Naval Postgraduate School's Simulations of Experiments and Efficient Designs (SEED) Lab at http://diana.or.nps.navy.mil/~susan/SeedLab/index.html, under "Software Downloads." As the parameter space you wish to search gets larger, these study designs become more and more important. A simple, gridded design will quickly outpace available computing resources, even for fairly small numbers of parameters. NOLH represents an alternative to gridded designs that still affords a statistically valid sample of the parameter space. More information about NOLH designs to explore high-dimensional simulations can be found at Lucas (2002) and at the SEED Lab Web site, under "Papers."

Once the study has been defined in either the Gridded tab or CSV_DOE tab, there are only a few last items to specify before one can begin the runs. These last items are specified in the Study Info tab. An example of this tab can be found in Figure 4. Here you may enter information about who is performing the study, as well as a narrative about the study. All of this information is optional. What are important on this tab are Model Info and Model Run Info. In these areas, you specify the model that OMD should use to run the program and the number of replicates to run for each parameter combination. The number of replicates to run is usually driven by how many processors you have on which to run the program, how long it takes to run your model, and what level of statistical significance you wish to achieve. In the past, we have used anything from 10 replicates for very cursory, fast analyses to 25,000 replicates when we are particularly concerned with very low base-rate phenomena.

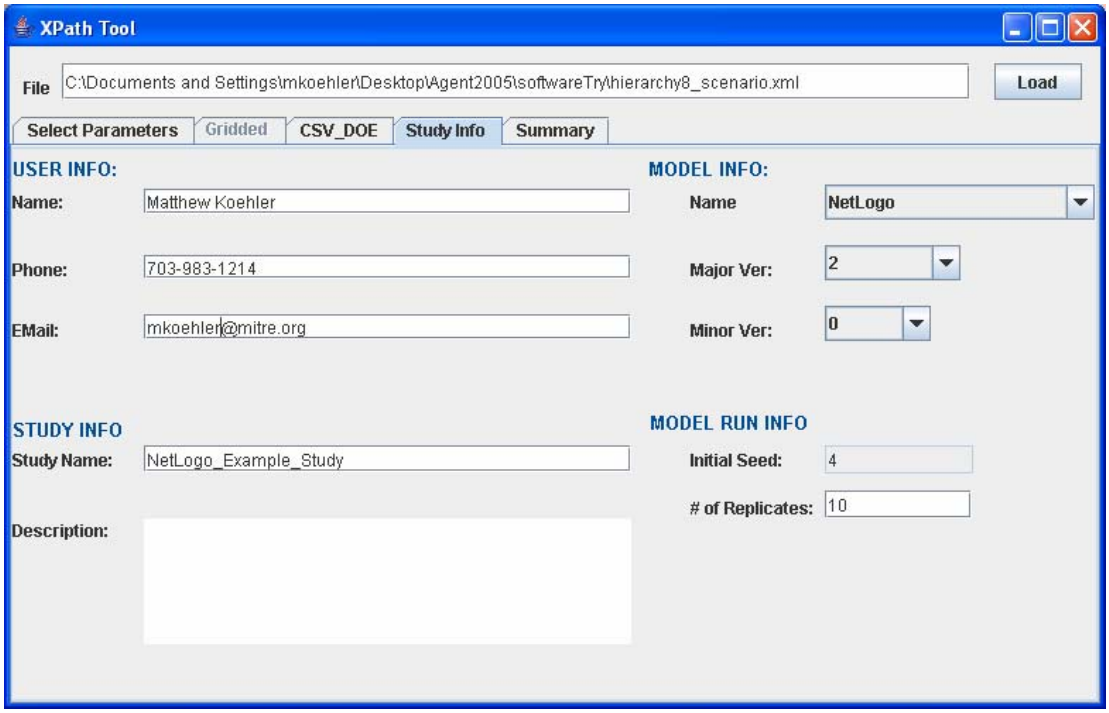

FIGURE 4 Study Info tab

The last step is to move to the Summary tab (Figure 5). Here you are presented with a summary of the information that you have entered thus far. At the bottom of the tab are two buttons. One button (Make Maui Study) is used to create a study.XML file for the Maui High Performance Computing Center; the other (Make OldMcData Study) is used to create a study.XML file for OMD. The only button that is of use in this case is the Make OldMcData Study button. This button will produce the necessary OMD XML file to run the experiment and place it in the same directory as the NetLogo XML file.

USING OLD MCDATA

Once you have created the OMD study.XML file, make sure everything is in the same directory as your NetLogo program. This includes, at a minimum, the .nlogo file and the study.XML file. If you imported a CSV file for the study design, that file must also be in the directory with the .nlogo and study.XML files.

Although this paper does not discuss the installation of OMD, the applications needed for OMD to function properly can be found in Table 1. Installation documentation for OMD can be found with the software, which is available from the authors and should soon be available on SourceForge.

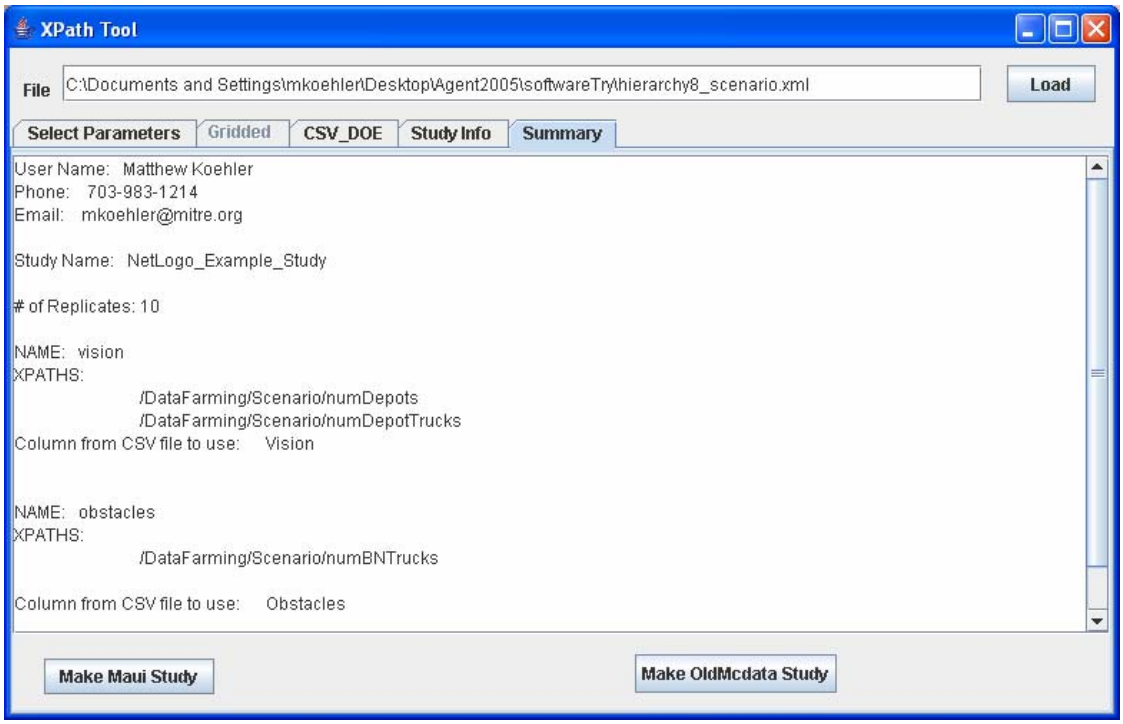

FIGURE 5 Summary tab

TABLE 1 Java applications required to run OldMcData

Application	Purpose	Jar files
Jade 2.5	Agent-based development environment	Base64.jar, iiop.jar, jadeTools.jar, jade.jar
Colt	Scientific code – use random number generators	colt1.0.2.jar
PES	Redirecting output	PES.jar
Xalan	XSLT processor for transforming XML – used by dom4j	xalan.jar
Xerces	XML parsing	Xerces.jar
Jakarta-oro-2.0.7	Regular expression pattern matching and processing	jakarta-oro-2.0.7.jar
NALEX	Natural algorithms, such as simulated annealing, genetic algorithms, and evolutionary programming	nalex1.0-20031119b306.jar
dom4j	XML parsing	dom4j-full.jar

Once OMD is installed on your computer, you will have the directory `oldmcdata` in your root directory. Place the NetLogo folder you created previously in the `test` directory found within the `oldmcdata` directory. In the following example, the NetLogo directory is called `NetLogoTest`. Once you have accomplished this, you should have the following directory tree: `c:/oldmcdata/test/NetLogoTest`. To run the experiment, open a command prompt window, navigate to the `oldmcdata` directory, and type in the following line:

```
oldmcdata.start c:/oldmcdata/test/NetLogoTest study.xml
```

The above is mostly for convenience and bookkeeping. You may put your NetLogo study directory anywhere you wish, thus making the command line statement:

```
oldmcdata.start <path to your study directory> <your study filename>
```

Unless you have manually changed the name of your study file that XStudy created, that last argument in the statement will remain `study.xml`.

Upon completion of the run, there will be three new folders in the `NetLogoTest` folder: `Excursions`, `Output`, and `Playback`. The `Excursions` folder contains the XML files used as input for each NetLogo run. The `Output` folder contains the output data that NetLogo created. The `Playback` folder should be empty and is created to hold output produced by some of the Project Albert models. At this point, you can use whatever tool you choose for analyzing the output data.

CONCLUSION

Project Albert has created a number of tools for running simple models for a large numbers of times in a cluster computing environment. Initially, these tools were specifically for Project Albert agent-based combat models. Recently, however, we have spent time trying to generalize this capability for use by the greater analytic and academic community. This is still very much a work in progress, and we would welcome collaboration with others as we continue to develop this capability and make it more accessible. This paper necessarily glossed over some details with regard to the process described; we encourage interested readers to contact the authors for more information and the latest versions of the software discussed herein. By November 15, 2005, all of the software discussed in this paper should be available on SourceForge. The Web site is http://sourceforge.net/projects/datafarm. The software is currently in an alpha form but will continue to be updated on the SourceForge site.

Although space limitations do not permit its detailed description, the reader should note that a similar XML framework has been developed for Repast J utilizing a custom controller that passes in a random seed and an XML input file. However, this system currently requires more customized code than the NetLogo version. As with the NetLogo case, the authors also wish to share this framework with interested members of the Repast community and will gladly share software and lessons learned with any interested party.

REFERENCES

Barry, P., and M. Koehler, 2004a, "Simulation in Context: Using Data Farming for Decision Support," in *Proceedings of the 2004 Winter Simulation Conference*, eds. R. Ingalls, M. Rossetti, J. Smith, and B. Peters, Institute of Electrical and Electronics Engineers, Piscataway, NJ; available at http://www.informs-sim.org/wsc04papers/101.pdf.

Barry, P., M. Koehler, and A. Forsyth, 2004b, "The Bigger Hammer Approach: Using Massively Parallel Computation to Address Low Base Rate Problems," in *Proceedings of the 2004 North American Computational Social and Organizational Science Conference;* available at http://www.casos.cs.cmu.edu/events/conferences/2004/2004_proceedings/Barry_Philip.pdf.

Brandstein, A., and G. Horne, 1998, "Data Farming: A Meta-Technique for Research in the 21st Century," in *Maneuver Warfare Science 1998*, eds. F. Hoffman and G. Horne, USMC Combat Development Command, Quantico, VA.

Horne, G., 2001, "Beyond Point Estimates: Operational Synthesis and Data Farming," in *Maneuver Warfare Science 2001*, eds. G. Horne and M. Leonardi, USMC Combat Development Command, Quantico, VA.

Lucas, T., S. Sanchez, Maj. L. Brown, and Maj. W. Vinyard, 2002, "Better Designs for High Dimensional Explorations of Distillations," in *Maneuver Warfare Science 2002*, eds. G. Horne and S. Johnson, USMC Project Albert, Quantico, VA.

Wilensky, U., 1999, *NetLogo. Center for Connected Learning and Computer-Based Modeling*, Northwestern University, Evanston, IL; available at: http://ccl.northwestern.edu/netlogo/.